AI Assisted Coding

Lab Assignment 6.3

Name : K. Charan Yadav

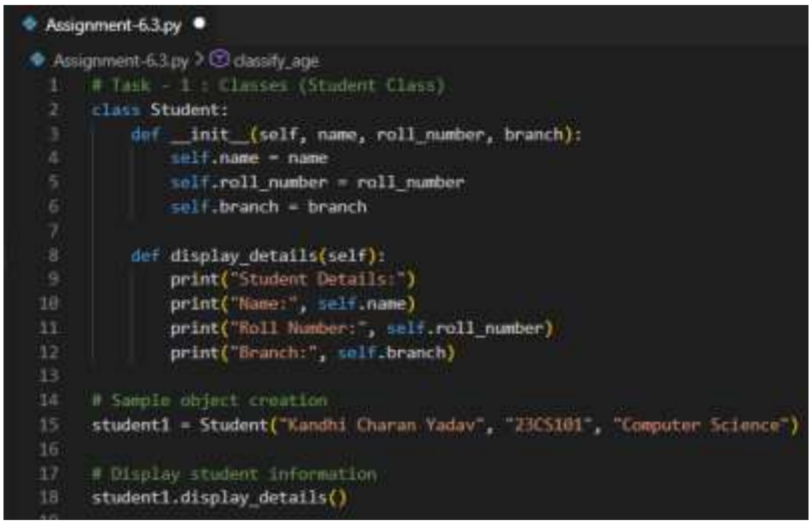
Hall Ticket no : 2303A52367

Batch No : 20

Task -1:

Prompt:

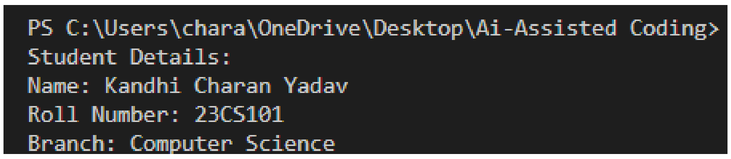
- Create a simple Python program using Object-Oriented Programming.
- Define a Student class with attributes name, roll_number, and branch.
- Include a constructor (__init__) to initialize these attributes.
- Add a method display_details() that prints the student's information clearly.
- Create a sample student object and call the display_details() method to show the output on the console.



Assignment-6.3.py
Assignment-6.3.py > classify_age

```python
# Task - 1 : Classes (Student Class)
class Student:
    def __init__(self, name, roll_number, branch):
        self.name = name
        self.roll_number = roll_number
        self.branch = branch

    def display_details(self):
        print("Student Details:")
        print("Name:", self.name)
        print("Roll Number:", self.roll_number)
        print("Branch:", self.branch)

# Sample object creation
student1 = Student("Kandhi Charan Yadav", "23CS101", "Computer Science")

# Display student information
student1.display_details()
```

OUTPUT :



```
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>
Student Details:
Name: Kandhi Charan Yadav
Roll Number: 23CS101
Branch: Computer Science
```

Justification:

AI assistance was used to generate a simple Student class demonstrating core OOP concepts such as class, constructor, and methods. The code correctly initializes student attributes and displays details in a clear format. It improves understanding of object creation and data encapsulation. The structure is readable and easy to extend.

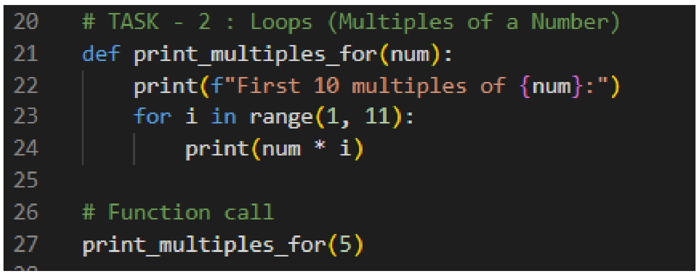
Task 2:

Prompt:

• Prompt the AI tool to generate a function that prints the first 10 multiples of a given number using a loop.

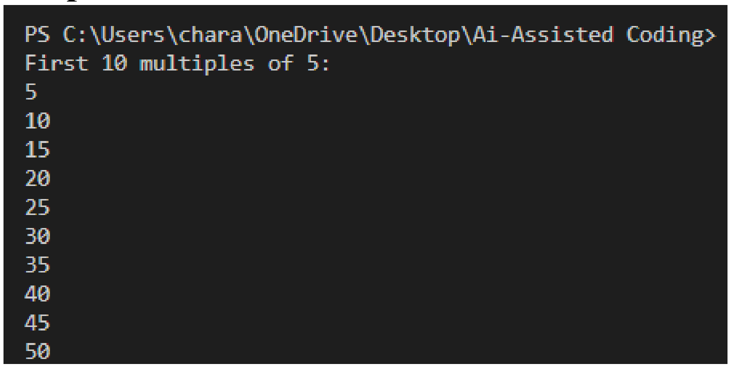
• Analyze the generated loop logic.

• Ask the AI to generate the same functionality using another controlled looping structure (e.g., while instead of for).



```python
# TASK - 2 : Loops (Multiples of a Number)
def print_multiples_for(num):
    print(f"First 10 multiples of {num}:")
    for i in range(1, 11):
        print(num * i)

# Function call
print_multiples_for(5)
```

Output:



```
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>
First 10 multiples of 5:
5
10
15
20
25
30
35
40
45
50
```

Justification:

AI generated solutions using both for and while loops to print the first 10 multiples of a number. This helped in understanding different loop control mechanisms. The logic was correct and avoided infinite loops. Comparing both approaches improved conceptual clarity.

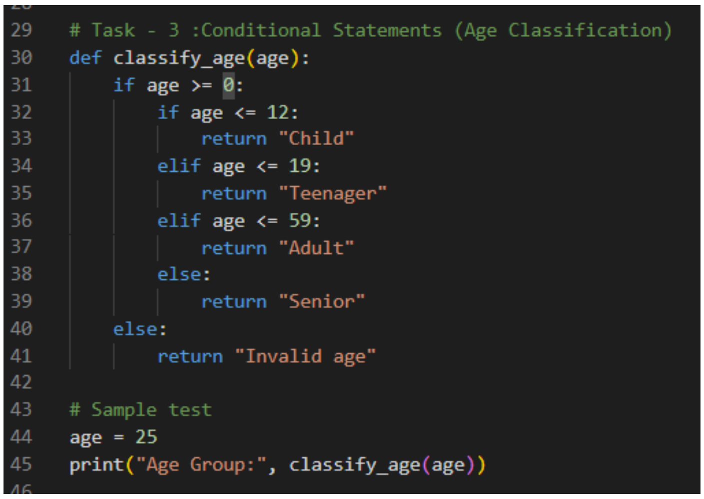
Task 3:

Prompt :

• Ask the AI tool to generate nested if-elif-else conditional statements to classify age groups (e.g., child, teenager, adult, senior).

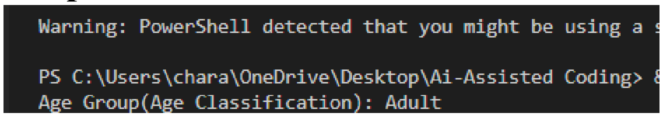
• Analyze the generated conditions and logic.

• Ask the AI to generate the same classification using alternative conditional structures (e.g., simplified conditions or dictionary-based logic).



```python
# Task - 3 :Conditional Statements (Age Classification)
def classify_age(age):
    if age >= 0:
        if age <= 12:
            return "Child"
        elif age <= 19:
            return "Teenager"
        elif age <= 59:
            return "Adult"
        else:
            return "Senior"
    else:
        return "Invalid age"

# Sample test
age = 25
print("Age Group:", classify_age(age))
```

Output :



```
Warning: PowerShell detected that you might be using a s

PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> &
Age Group(Age Classification): Adult
```

Justification:

AI created nested and simplified conditional statements to classify age groups accurately. The conditions were logically ordered and non-overlapping. Alternative approaches improved readability and scalability. This task strengthened decision-making concepts.

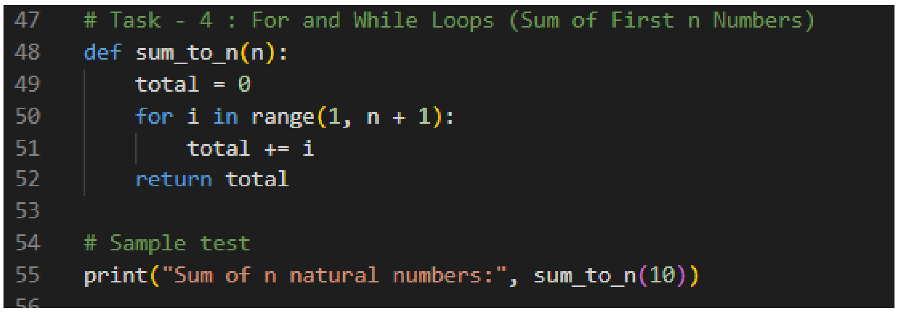
Task 4 :

Prompt :

• Use AI assistance to generate a sum_to_n() function using a for loop.

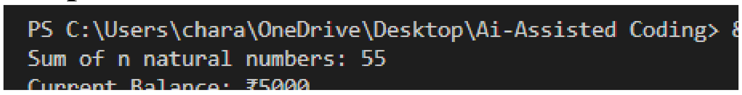
• Analyze the generated code.

• Ask the AI to suggest an alternative implementation using a while loop or a mathematical formula.



```python
# Task - 4 : For and While Loops (Sum of First n Numbers)
def sum_to_n(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

# Sample test
print("Sum of n natural numbers:", sum_to_n(10))
```

Output :



```
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> &
Sum of n natural numbers: 55
Current Balance: ₹5000
```

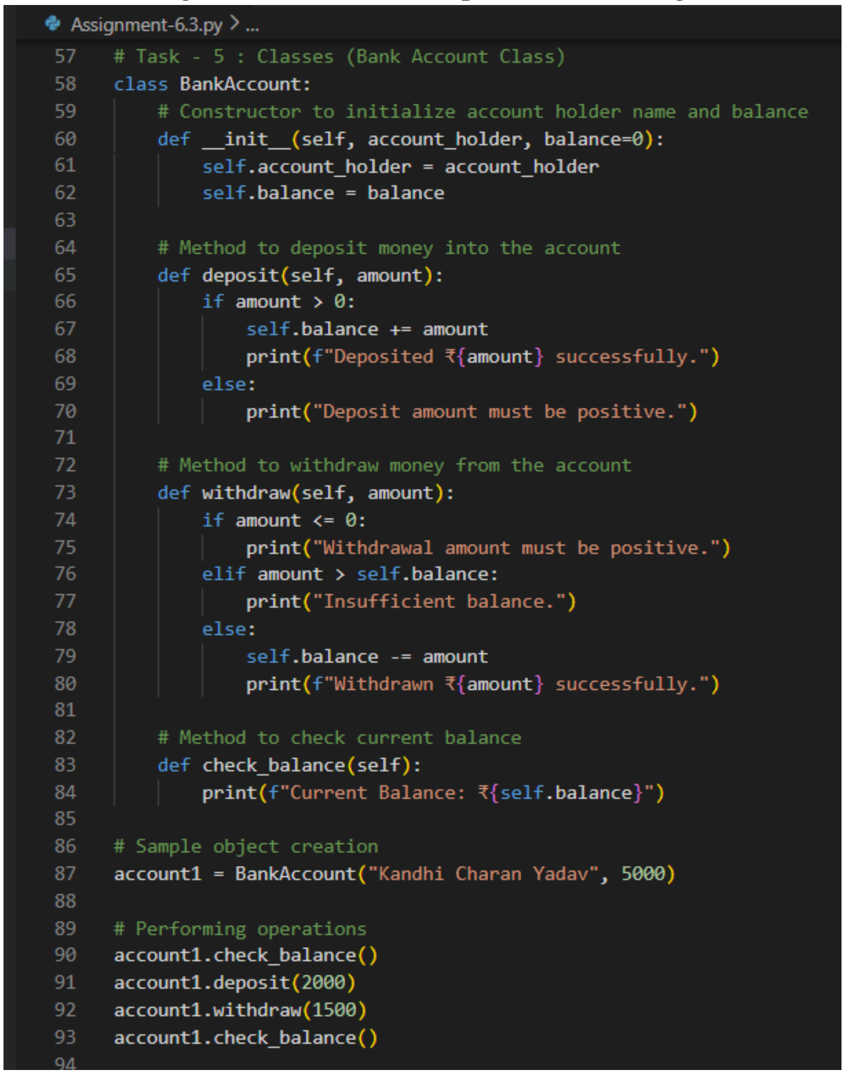
Justification:

sum_to_n() Function AI provided loop-based and formula-based implementations to calculate the sum of numbers up to n. This comparison highlighted efficiency and time complexity differences. The formula-based solution showed optimal performance. The task reinforced algorithm optimization skills.

Task 5 :

Prompt :

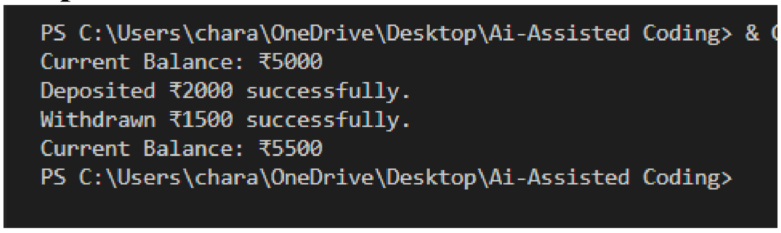
• Use AI tools to generate a Bank Account class with methods such as deposit(), withdraw(), and check_balance().
• Analyze the AI-generated class structure and logic.
• Add meaningful comments and explain the working of the code.



Assignment-6.3.py > ...

```python
# Task - 5 : Classes (Bank Account Class)
class BankAccount:
    # Constructor to initialize account holder name and balance
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    # Method to deposit money into the account
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited ₹{amount} successfully.")
        else:
            print("Deposit amount must be positive.")

    # Method to withdraw money from the account
    def withdraw(self, amount):
        if amount <= 0:
            print("Withdrawal amount must be positive.")
        elif amount > self.balance:
            print("Insufficient balance.")
        else:
            self.balance -= amount
            print(f"Withdrawn ₹{amount} successfully.")

    # Method to check current balance
    def check_balance(self):
        print(f"Current Balance: ₹{self.balance}")

# Sample object creation
account1 = BankAccount("Kandhi Charan Yadav", 5000)

# Performing operations
account1.check_balance()
account1.deposit(2000)
account1.withdraw(1500)
account1.check_balance()
```

Output :



```
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding> & C
Current Balance: ₹5000
Deposited ₹2000 successfully.
Withdrawn ₹1500 successfully.
Current Balance: ₹5500
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>
```

Justification:

AI generated a Bank Account class representing real-world banking operations using OOP. The methods included proper validation for deposit and withdrawal. Meaningful comments improved code readability. The design is logical, secure, and reusable.